NATIVE XML : XPATH AND XQUERY, ARE YOU READY?

KURT STRUYF
INFOCURA

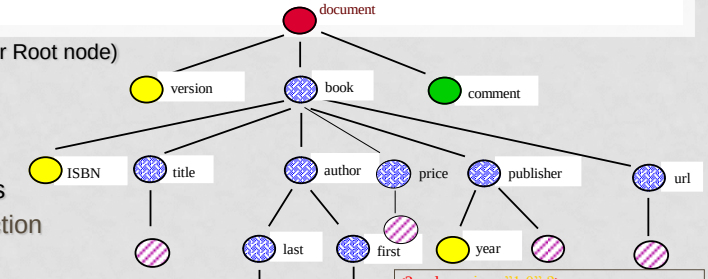
WE CARE ABOUT YOUR INFORMATION

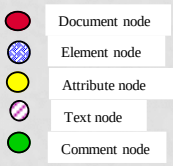
XML

- eXtensible Markup Language
  - Structured data in a flat text
  - Self describing
  - Hierarchical in nature
  - Flexible in design
  - Machine and man-readable
  - De facto data exchange standard

TYPES OF XML NODES

Document node (or Root node)
Element nodes
Text nodes
Attribute nodes
Namespace nodes
Processing instruction nodes
Comment nodes






```
<?xml version="1.0" ?>
  <book ISBN="0439708184">
   <title>
      Harry Potter and the sorcerer's stone
   </title>
   <author>
      <last>Rowling</last>
      <first>J.K.</first>
   </author>
   <publisher year="1999">
      Scholastic Paperbacks
   </publisher>
   <price>19.85</price>
   <URL>
      http://harrypotter.bloomsbury.com
   </URL>
  </book>
<!-- comment at end  -->
```

WHAT IS WELL-FORMED?

- An XML document is well-formed, if…
  - It complies with the syntax rules
  - It can be parsed by an XML parser without error
- Some rules (XML manual for full list):
  - Has ONE root element

```
<a> 123</a>
<b> abc</b>
```

```
<c>
<a> 123</a>
<b> abc</b>
</c>
```

  - Matching open/close tags
    - Elements correctly nested
    - Tags are case sensitive

```
<c>
<a>
<b> abc</A>
</b>
```

```
<c>
<a>
<b> abc</b>
</a>
</c>
```

  - Attributes are quoted

```
<b id= 5>bob </b>
```

```
<b id= ”5 ”>bob </b>
```

WHAT IF NOT WELL-FORMED ?

```
<award>
 <name> dellacort award </awardname>
 <year> 2007 </year>
</award>
```

DSNT408I SQLCODE = **-20398**, ERROR: ERROR
ENCOUNTERED DURING XML PARSING AT
LOCATION 1764 Element end tag name does not match start tag

WHAT IS VALID ?

- An XML document is valid, if…
  - It is well-formed AND
  - It complies with a specific XML schema or DTD (=document type definition)
- XML schema or DTD
  defines a specific XML document structure
- Validation can be done by XML parsers

more about this later

HOW TO STORE IT ?

- Option 1 : Shred it
  - Convert your XML data to relational
    - Might require many relational tables
    - Lose XML flexibility
    - Reading all data, means joining many tables
- Option 2 : Store it
  - As a character string
  - As a CLOB
  - Native XML

HOW TO STORE IT ?

- Native XML :
  - New data type in DB2
  - Advantages
    - More flexible use of build in XML-functions
    - XML document validation
    - Indexing on XML-columns possible
  - Disadvantages:
    - DBA need XML storage/usage knowledge

HOW TO STORE IT ?

- Native XML :

```
CREATE DATABASE KSTDB;

    CREATE TABLESPACE KSTTS
    IN KSTDB;

CREATE TABLE KST.CUSTOMER
(CUSTID   INTEGER,
 INFO     XML)
 DOCID column is implicitly added

IN KSTDB.KSTTS;
```

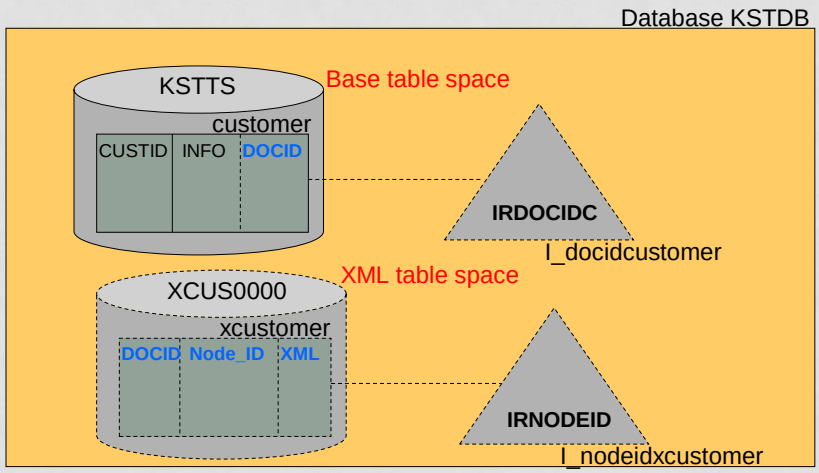
```
CREATE TABLESPACE KSTBOTS
     IN KSTDB;

CREATE TABLE KST.BOOKSTORE
(STOREID   INTEGER,
 STOCK     XML)
 DOCID column is implicitly added

IN KSTDB.KSTBOTS;
```

XML COLUMN – IMPLICITLY/EXPLICITLY CREATED OBJECTS

```
CREATE TABLE KST.CUSTOMER(CUSTID   INTEGER, INFO     XML) IN KSTDB.KSTTS;
```



DISPLAY DB(KSTDB)

```
DSNT360I  - ************************************
DSNT361I  - *  DISPLAY DATABASE SUMMARY
            *    GLOBAL
DSNT360I  - ************************************
DSNT362I  -     DATABASE = KSTDB  STATUS = RW
                DBD LENGTH = 4028
DSNT397I  -
NAME     TYPE PART  STATUS            PHYERRLO PHYERRHI CATALOG  PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
KSTTS    TS         RW
XCUS0000 XS    0001 RW
XCUS0000 XS         RW
IRDOCIDC IX         RW
IRNODEID IX   L0001 RW
IRNODEID IX    L*   RW
******* DISPLAY OF DATABASE KSTDB    ENDED      **********************
```

VALIDATE YOUR XML

- DB2 supports XML schema's to validate XML documents beyond « well formed »
- Using schema's to check:
  - Data type checking of elements
  - Length and pattern checking for strings
  - « must have/not null » checking for elements
  - Name space checking
  - Etc

HOW TO IMPLEMENT XML SCHEMAS

- Preferred method (only in DB2 10)
  - Define the schema at a table level similar to « check constraint »

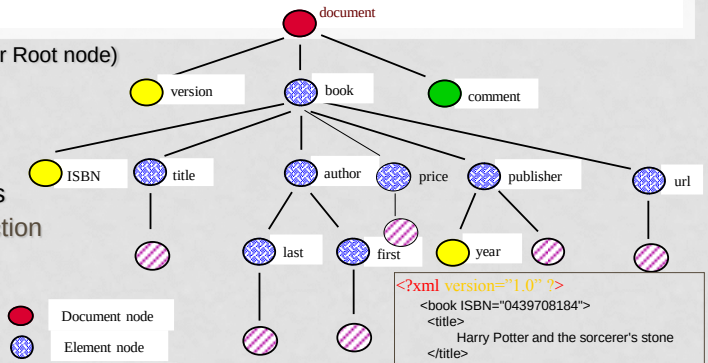
```
CREATE TABLE KST.CUSTOMER
(CUSTID    INTEGER,
 INFO      XML (XMLSCHEMA SYSXSR.KURT))
 DOCID column is implicitly added
IN KSTDB.KSTTS;
```

- Alternative method (available DB2 9 & 10)
  - Use DSN_XMLVALIDATE function at every insert/update/delete

```
INSERT INTO KST.CUSTOMER VALUES (102,
        SYSIBM.DSN_XMLVALIDATE(:xmldoc, 'SYSXSR.KURT'));
```

TYPES OF XML NODES

Document node (or Root node)
Element nodes
Text nodes
Attribute nodes
Namespace nodes
Processing instruction nodes
Comment nodes




```
<?xml version="1.0" ?>
  <book ISBN="0439708184">
    <title>
        Harry Potter and the sorcerer's stone
    </title>
    <author>
        <last>Rowling</last>
        <first>J.K.</first>
    </author>
    <publisher year="1999">
        Scholastic Paperbacks
    </publisher>
    <price>19.85</price>
    <URL>
        http://harrypotter.bloomsbury.com
    </URL>
  </book>
<!-- comment at end  -->
```

WHAT IS A VALIDATION SCHEMA ?

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://posample.org"
xmlns:xsd="http://www.w3.org/2013/XMLSchema"
elementFormDefault="qualified">
<xs:element name="books">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="book" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="title" type="xs:string" />
      <xs:element name="Author" maxOccurs="unbounde
       <xs:complexType>
        <xs:sequence>
         <xs:element name="last" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="first" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
     <xs:element name="publisher" maxOccurs="1">
      <xs:complexType>
           <xs:simpleContent>
       <xs:extension base="xs:string">
        <xs:attribute name="year" type="xs:integer" />
       </xs:extension>
      </xs:simpleContent>
     </xs:complexType>
    </xs:element>
    <xs:element name="price" type="xs:decimal" minOccurs="1" maxOccurs="1"/>
   <xs:element name="awards" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
           <xs:element name="award" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
       <xs:element name="year" type="xs:integer" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>  </xs:complexType>      </xs:element>
    </xs:complexType>  </xs:element>
   </xs:sequence> </xs:complexType>
  <xs:attribute name="isbn" type="xs:integer" />
 </xs:element>  </xs:sequence> </xs:complexType> </xs:element></xs:schema
```

How to introduce this to DB2 ?

Two stored procedures

1. SYSPROC.XSR_REGISTER ('SYSXSR','KURT',:LOC,:CONTENT,:PROPERTY)
2. CALL SYSPROC.XSR_COMPLETE

Assume no awards

```
<book ISBN="0439708184">
 <title>
      Harry Potter and the sorcerer's stone
 </title>
 <author>
      <last>Rowling</last>
      <first>J.K.</first>
 </author>
 <publisher year="1999">
      Scholastic Paperbacks
 </publisher>
 <price>19.85</price>
</book>
```

HOW DO YOU KNOW YOUR XML IS VALIDATED

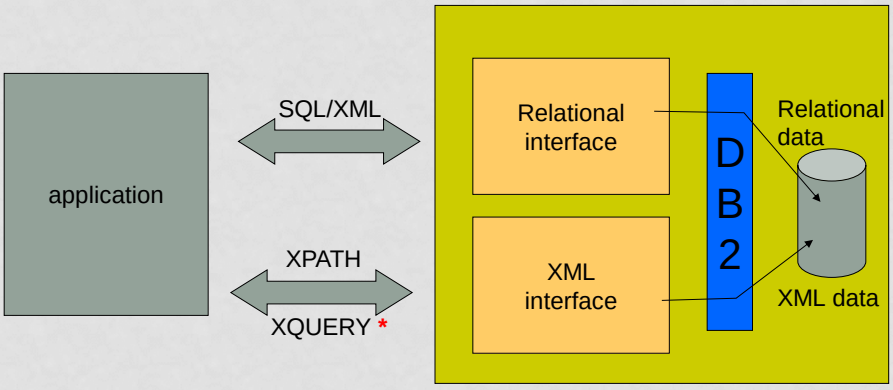
```
SELECT COUNT(*)
FROM KST.BOOKSTORE
WHERE XMLXSROBJECTID(STOCK)=0
```

Gives you how many XML documents have not been validated

```
UPDATE KST.BOOKSTORE
SET STOCK = DSN_XMLVALIDATE(STOCK, 'SYSXSR.KURT')
 WHERE XMLXSROBJECTID(STOCK)=0
```

Validates all XML documents with schema SYSXSR.KURT that have not been validated

RELATIONAL OR XML?



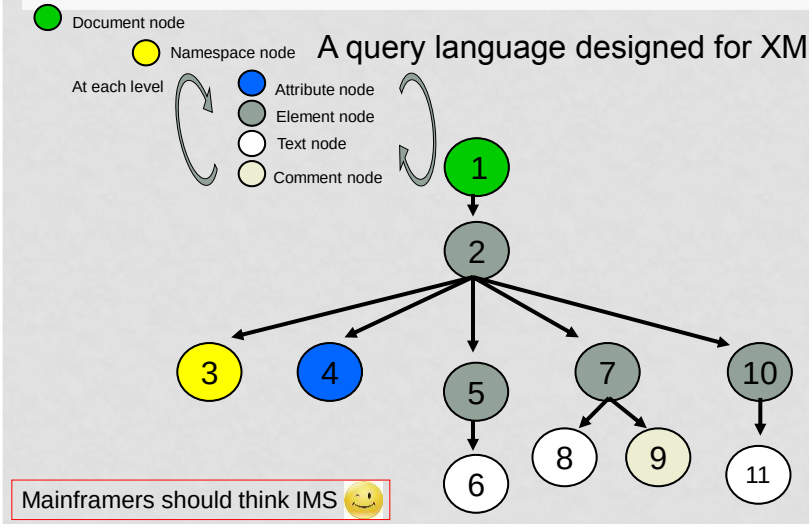
* DB2 10 for z/OS after GA applying PTF'S UK72208 & UK73139

XML FUNCTIONS DELIVERED WITH DB2

Two major types:
1. Functions to retrieve data from an XML column
   - Xpath
   - Xquery
2. Functions to build XML data

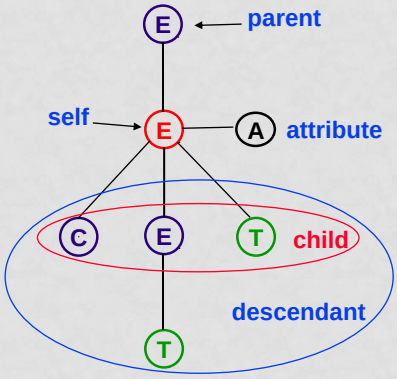
Let's have a look at XPath!

XML NAVIGATION



SUPPORTED AXES

- Forward Axis:
  - child *(default)*
  - descendant
  - attribute (@)
  - self (.)
  - descendant-or-self (//=/descendant-or-self:node()/)
- Reverse Axis:
  - parent (..)



E : element
A : attribute
C : comment
T: Text

XPATH SYNTAX

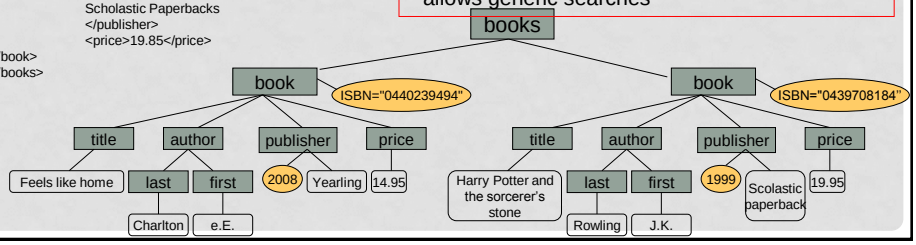
```
<books>
<book ISBN="0440239494">
        <title>feels like home</title>
        <author>
            <last>Charlton</last>
            <first>e.E.</first>
        </author>
        <publisher year="2008">
            Yearling
        </publisher>
        <price>9.85</price>
</book>
<book ISBN="0439708184">
        <title>Harry Potter and the sorcerer's stone</title>
        <author>
            <last>Rowling</last>
            <first>J.K.</first>
        </author>
        <publisher year="1999">
        Scholastic Paperbacks
        </publisher>
        <price>19.85</price>
</book>
</books>
```

Navigate using the node's path

/books
/books/book
/books/book/@ISBN
/books/book/title
/books/book/author
/books/book/publisher/@year
…

/books/*/price
//price
/books//@* or //@*

/ means root or step
// means one or more steps (decendant)
@ indicates an attribute
* allows generic searches



XPATH EXAMPLES

```
<books>
<book ISBN="0440239494">
        <title>feels like home</title>
        <author>
            <last>Charlton</last>
            <first>e.E.</first>
        </author>
        <publisher year="2008">
            Yearling
        </publisher>
        <price>9.85</price>
</book>
<book ISBN="0439708184">
        <title>Harry Potter and the sorcerer's stone</title>
        <author>
            <last>Rowling</last>
            <first>J.K.</first>
        </author>
        <publisher year="1999">
        Scholastic Paperbacks
        </publisher>
        <price>19.85</price>
</book>
</books>
```

/books/book/@ISBN	0440239494 0439708184
/books/book/title //title	<title>feels like home </title> <title>harry potter and … </title>
//price/text() ➔ **just the text, no tags**	9.85 19.85
/books/*/@year ➔ **generic searches**	2008 1999
/books/book**[first="e.E."]**/title ➔**search/where criteria**	<title>feels like home </title>
//book[arg1 or arg2] ➔ **multiple search arguments possible**	

Remarks :

1. for performance reasons **best** use fully qualified paths if possible
2. Search criteria are **always** between [ ]

MORE XPATH SEARCH EXAMPLES

```
<books>
<book ISBN="0440239494">
        <title>feels like home</title>
        <author>
            <last>Charlton</last>
            <first>e.E.</first>
        </author>
        <publisher year="2008">
            Yearling
        </publisher>
        <price>9.85</price>
</book>
<book ISBN="0439708184">
        <title>Harry Potter and the sorcerer's stone</title>
        <author>
            <last>Rowling</last>
            <first>J.K.</first>
        </author>
        <publisher year="1999">
        Scholastic Paperbacks
        </publisher>
        <price>19.85</price>
</book>
</books>
```

/books/book[first="e.E."]/title ➔search/where criteria	<title>feels like home </title>
/books/book/title**[../first="e.E."]** **➔parent search/where criteria**	<title>feels like home </title>
/books/book[@ISBN="0439708184"]/@ISBN	0439708184
/books/book/@ISBN[.="0439708184"] **➔ Self search criteria**	0439708184
/books/book**[title='home"]**/title **➔wrong search criteria**	No result
/books/book **[contains(title,"home")]**/title **➔search criteria equivalent to like**	<title>feels like home </title>

Remarks :

1. ' **.** ' points at the current context
2. ' **..** ' points at the parent context

NAMESPACES

- Naming conflicts
  In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

Example 1 : HTML table

```
<table>
     <tr>
        <td>Apples</td>
        <td>Bananas</td>
     </tr>
</table>
```

Example 2 : table in furniture store XML

```
<table>
   <description>
     white oak coffee table
   </description>
   <width>80</width>
   <length>100</length>
   <height>60</height>
</table>
```

- Merge of these XML documents would result in a naming conflict.
  - <table> element exists but with different content and meaning
  - XML parser will not know how to handle these differences

NAMESPACES

- Name conflict can be solved by a prefix
- When using a prefix, the namespace attribute (xmlns) must be defined in the start tag of element

```
<t:table xmlns:t=''http://www.testshop.com/html4''>
     <t:tr>
        <t:td>Apples</t:td>
        <t:td>Bananas</t:td>
     </t:tr>
</t:table>
```

- When a namespace is defined for an element, all child elements are associated with the same namespace.
- The namespace URL is not used by the parser to look up information. The purpose is to give the namespace a unique name
- Path becomes ➔ /t:table/t:tr/t:td

XMLQUERY

- XMLQUERY allows you to extract partial information from an XML-document.

```
SELECT STOREID, XMLSERIALIZE(STOCK as char large object) as ''xml-value''
FROM KST.BOOKSTORE
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <books><book ISBN="0440239494"><title>feels like home</title><aut...
 2  <books><book ISBN="0440239494"><title>feels like home</title><aut...
 5  <books><book ISBN="0440239496"><title>feels like home</title><aut...
...
```

```
SELECT STOREID, XMLSERIALIZE(XMLQUERY ('//books/book/title' passing STOCK )
                as char large object)  as ''xml-value''
FROM KST.BOOKSTORE
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <title>feels like home</title><title>harry potter and the sorc...
 2  <title>feels like home</title><title>harry potter and the sorc...
 5  <title>feels like home</title><title>harry potter and the sorc...
...
```

XMLQUERY

- XMLQUERY **does not** filter any rows

```
  SELECT STOREID,
  XMLSERIALIZE(XMLQUERY ('/books/book[@ISBN="0440239494"]/title'
  passing STOCK ) as char large object) as "xml-value"
  FROM KST.BOOKSTORE
 ;
---------+---------+---------+-------
    STOREID   xml-value
---------+---------+---------+-------
         1  <title>feels like home</title>
         2  <title>feels like home</title>
         5
...
```

Only the requested information is returned matching the search criteria

No XML-value to match the search criteria but the row is returned

XMLEXISTS

- XMLEXISTS filters rows, but does not limit the XML-value returned

```
SELECT STOREID,
 XMLSERIALIZE(XMLQUERY ('/books/book/title'  passing STOCK ) as char large object)
 FROM KST.BOOKSTORE
 WHERE XMLEXISTS('$x/books/book[@ISBN="0440239494"]'
 passing by ref
 STOCK as "x")
;
---------+---------+---------+--------
   STOREID
---------+---------+---------+---------+---------+---------+---------+---------+
         1  <title>feels like home</title><title>prizefighter en mi casa</title...
         2  <title>feels like home</title><title>prizefighter en mi casa</title...
```

XMLquery determines what is in result ALL titles are shown not just those that « meet XMLEXISTS clause »

In result only rows for which the where XMLEXISTS clause is TRUE

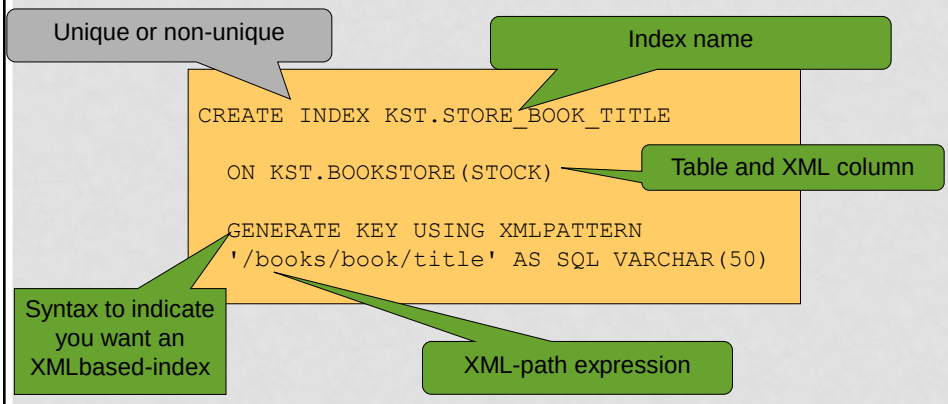
XMLEXISTS & XMLQUERY

```
SELECT STOREID,
 XMLSERIALIZE(XMLQUERY ('/books/book[@ISBN="0440239494"]/title'
 passing STOCK ) as char large object)
 FROM KST.BOOKSTORE
 WHERE XMLEXISTS('$x/books/book[@ISBN="0440239494"]'
 passing by ref
 STOCK as "x")
;
---------+---------+---------+---------+---------+---------+-------
   STOREID
---------+---------+---------+---------+---------+---------+-------
         1  <title>feels like home</title>
         2  <title>feels like home</title>
```

- Only XMLEXISTS clause can use an INDEX

XML INDEX

- Only XMLEXISTS clause can use an INDEX



XML FUNCTIONS DELIVERED WITH DB2

Two major types:
1. Functions to retrieve data from an XML column
   - Xpath
   - Xquery
2. Functions to build XML data

Let's have a look at XQuery!

XMLQUERY USING XQUERY

- XQUERY shows simularities with Xpath

```
 SELECT XMLQUERY( '/BOOKS/BOOK/TITLE' PASSING STOCK )
 FROM KST.BOOKSTORE ;
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <title>feels like home</title><title>harry potter and the sorc...
 2  <title>feels like home</title><title>harry potter and the sorc...
 5  <title>feels like home</title><title>harry potter and the sorc...
...
```

Xpath

```
  SELECT XMLQUERY('for $I in  $O/BOOKS/BOOK/TITLE
     return $I' PASSING STOCK AS "O" )
 FROM KST.BOOKSTORE ; ;
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <title>feels like home</title><title>harry potter and the sorc...
 2  <title>feels like home</title><title>harry potter and the sorc...
 5  <title>feels like home</title><title>harry potter and the sorc...
...
```

Xquery

XQUERY COMPARED TO XPATH

Xquery:

- is based on Xpath expression
- Allows more functional complexity
  - Mathematical expression
  - Result transformation
  - If-then-else constructs
- Is to XML what SQL is to relational databases using **FLWOR** (a.k.a. "FLOWER") expressions

DB2 10 for z/OS after GA applying PTF'S UK72208 & UK73139

FLWOR EXPRESSION

FLWOR stands for:

For : Defines the input/source
Let : allows for variable declaration and gives it a value
Where : search condition
Order by : sort the output
Return : defines the output of the FLWOR expression

FLWOR

```
for $v in $doc//video
where $v/year = 1999
order by $v/title ascending
return $v/title
```

SQL

```
SELECT v.title
FROM video v
WHERE v.year = 1999
ORDER BY v.title ascending
```

34

MOST SIMPLE XQUERY EXAMPLE

```
  SELECT XMLQUERY('for $I in  $O/BOOKS/BOOK/TITLE
     return $I' PASSING STOCK AS "O" )
 FROM KST.BOOKSTORE ; ;
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <title>feels like home</title><title>harry potter and the sorc...
 2  <title>feels like home</title><title>harry potter and the sorc...
 5  <title>feels like home</title><title>harry potter and the sorc...
...
```

Xquery

In it most simple form an Xquery contains a
for clause
and a
return clause

Mainframers should be aware of case sensitivity !!

FLWOR KEYWORDS ARE LOWER CASE

```
 SELECT XMLQUERY(
       'FOR $I IN  $O/BOOKS/BOOK/TITLE
       RETURN $I'
       PASSING STOCK AS "O" )
   FROM KST.BOOKSTORE ;
---------+---------+---------+---------+---------+---------+---------+---
DSNT408I SQLCODE = -16002, ERROR:  AN XQUERY EXPRESSION HAS AN UNEXPECTED
         TOKEN $ FOLLOWING FOR. EXPECTED TOKENS MAY INCLUDE: . ERROR
         QNAME=err:XPST0003
```

```
  SELECT XMLQUERY('for $I in  $O/BOOKS/BOOK/TITLE
     return $I' PASSING STOCK AS "O" )
 FROM KST.BOOKSTORE ; ;
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <title>feels like home</title><title>harry potter and the sorc...
 2  <title>feels like home</title><title>harry potter and the sorc...
 5  <title>feels like home</title><title>harry potter and the sorc...
...
```

SIMPLE LET EXPRESSION

```
  SELECT XMLQUERY('for $I in  $O/BOOKS/BOOK/TITLE
     return $I' PASSING STOCK AS "O" )
 FROM KST.BOOKSTORE ; ;
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <title>feels like home</title><title>harry potter and the sorc...
 2  <title>feels like home</title><title>harry potter and the sorc...
 5  <title>feels like home</title><title>harry potter and the sorc...
...
```

```
  SELECT XMLQUERY('for $J in  $O/BOOKS/BOOK
     let $I := $J/TITLE
     return $I' PASSING STOCK AS "O" )
 FROM KST.BOOKSTORE ; ;
---------+---------+---------+---------+---------+---------+------
    STOREID  xml-value
---------+---------+---------+---------+---------+---------+------
 1  <title>feels like home</title><title>harry potter and the sorc...
 2  <title>feels like home</title><title>harry potter and the sorc...
 5  <title>feels like home</title><title>harry potter and the sorc...
...
```

BIT MORE ADVANCED

```
 SELECT XMLSERIALIZE(XMLQUERY(
      'for $J in  $O/BOOKS/BOOK
       let $I := $J/TITLE,
           $A := $J/AWARDS/AWARD/NAME,
           $Q := fn:count($A)
      where $Q > 0
  return $I '
     PASSING STOCK AS "O" ) AS CLOB)
  FROM KST.BOOKSTORE ;


---------+---------+---------+---------
xml-value
---------+---------+---------+---------+---------+---------+------
<TITLE>PRIZEFIGHTER EN MI CASA</TITLE>
<TITLE>PRIZEFIGHTER EN MI CASA</TITLE>
<TITLE>PRIZEFIGHTER EN MI CASA</TITLE>
...
```

1 let clause, variables seperated by “,”

Fn: ➔ calls a function

Book must have at least 1 award

Straight forward result

COMPUTED VALUES AS RESULT

```
SELECT XMLSERIALIZE(XMLQUERY(
     'for $J in  $O/BOOKS/BOOK
      let $I := $J/TITLE,
          $A := $J/AWARDS/AWARD/NAME,
          $Q := fn:count($A)
     where $Q > 0
  return  <book> {$I}
            <awards> {$Q}
              <awardname> {fn:data($A)} </awardname>
            </awards>
          </book>'

     PASSING STOCK AS "O" ) AS CLOB)
  FROM KST.BOOKSTORE
WHERE STOREID = 1

---------+---------+---------+---------+---------+---------+------
xml-value
---------+---------+---------+---------+---------+---------+------
<book><TITLE>PRIZEFIGHTER EN MI CASA</TITLE><awards>2<awardname> DELLACORT AWARD...
```

Compute values between {}

Function to just have data

XQUERY FUNCTION EXAMPLES

fn:avg() : to calculate the average value
fn:compare(string1,string2) : comparing two strings
fn:concat(string1,string2..) : concatenating strings
fn:matches(source,pattern) : checks if source matches a pattern
example: check if a value is a 10digit number starting with 4
fn:matches($y, "^4[0-9]{9}$")
fn:replace(source,pattern) : replaces certain values
example: strip dashes from value
… **fn:replace($x, "-", "")**
Many more

DB2 10 for z/OS :
PureXML guide : Chapter 12
SC19-2981-05

SYNTAX (TO CONFUSE THE RUSSIANS)

^ matches the start of the string, while $ matches the end of the string

fn:matches($y, "^4[0-9]{9}$")

BUT

Within [ ] the ^ stands for negation e.g. non digits will be removed

fn:replace($x, "[^0-9]+", "")

XML FUNCTIONS DELIVERED WITH DB2

Two major types:

1. Functions to retrieve data from an XML column
   - Xpath
   - Xquery
2. **Functions to build XML data a.k.a XML constructs**

XML TABLE EXPRESSION

- Generates a relational « table » from an XML document

```
SELECT X.*
FROM CUSTOMER,
XMLTABLE (XMLNAMESPACES(DEFAULT 'http://posample.org'),
'//customerinfo'
PASSING CUSTOMER.INFO
COLUMNS "CUSTNAME" VARCHAR(30) PATH 'name',
        "addr/city" VARCHAR(30) PATH 'address') X
ORDER BY X.CUSTNAME
```

- Allowing SQL statements against an XML document

XML CONSTRUCT FUNCTIONS

- XMLELEMENT :Allows for building XML data
  function returns an XML value that is an XML element node.
- XMLATTRIBUTE : Allows for adding attributes to an XMLELEMENT node
  The XMLATTRIBUTES function constructs XML attributes from the arguments
- XMLDOCUMENT :
  The XMLDOCUMENT function returns
  - an XML value with a single document node.
  - zero or more nodes as its children.
  - The content of the generated XML document node is specified by a list of expressions.

XML CONSTRUCT FUNCTIONS

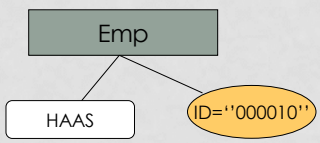
- XMLPARSE
  converts XML text to XML value
- XMLSERIALIZE
  converts XML to non-XML
- XMLQUERY
  executes an XPath expression against XML value
- XMLCAST
  casts XML to other types and vice versa
- XMLEXISTS
  predicate that allows row filtering base on XPath expression
- XMLFOREST
  returns an XML value that is a sequence of XML element nodes.

AND many more

XMLELEMENT, XMLATTRIBUTES,XMLCOMMENT, XMLDOCUMENT

```
INSERT INTO KST.CUSTOMER
VALUES('123', (SELECT XMLDOCUMENT (
                        XMLELEMENT (NAME "Emp"
                                    ,XMLATTRIBUTES(E.EMPNO as "ID")
                                    ,E.LASTNAME),
                        XMLCOMMENT ('This is just a simple example')
                                    )
                    FROM KST.EMP E
                    WHERE E.EMPNO = '000010'
                )
        ) ;
```

<Emp ID="000010">HAAS</Emp><!--This is just a simple example--> (on retrieval)



XMLELEMENT WITHOUT XMLDOCUMENT

```
SELECT
    XMLELEMENT (NAME "EMP", XMLATTRIBUTES(E.EMPNO AS ID)
                , E.LASTNAME
                ),
    XMLELEMENT (NAME "FIRST", E.FIRSTNME
                ),
    XMLCOMMENT ('THIS IS JUST A SIMPLE EXAMPLE')
FROM V910.EMP E
    WHERE E.EMPNO = '000010'
 ;
```

```
<?xml version="1.0" encoding="IBM500"?><EMP ID="000010">HAAS</EMP>  <?xml version="1.0"
encoding="IBM500"?><FIRST>CHRISTINE</FIRST>      <?xml version="1.0"
encoding="IBM500"?><!--THIS IS JUST A SIMPLE EXAMPLE-->
```

Returns 3 "columns" each containing an XML-document

XMLELEMENT WITH XMLDOCUMENT

```
  SELECT XMLDOCUMENT (
    XMLELEMENT (NAME "EMP", XMLATTRIBUTES(E.EMPNO AS ID)
                , E.LASTNAME
            ),
    XMLELEMENT (NAME "FIRST", E.FIRSTNME
            ),
    XMLCOMMENT ('THIS IS JUST A SIMPLE EXAMPLE')
            )
 FROM V910.EMP E
    WHERE E.EMPNO = '000010'
```

```
<?xml version="1.0" encoding="IBM500"?><EMP
ID="000010">HAAS</EMP><FIRST>CHRISTINE</FIRST><!--THIS IS JUST A SIMPLE EXAMPLE-->
```

Returns 1 column containing one XML-document

WHERE DO YOU XMLCOMMENT?

```
SELECT XMLDOCUMENT (
  XMLELEMENT (NAME "EMP", XMLATTRIBUTES(E.EMPNO AS ID)
                , E.LASTNAME
            ),
  XMLELEMENT (  NAME "FIRST", E.FIRSTNME,
                XMLCOMMENT ('FIRSTNME EXAMPLE')
            ),
  XMLCOMMENT ('THIS IS JUST A SIMPLE EXAMPLE')
            )
 FROM V910.EMP E
   WHERE E.EMPNO = '000010'
```

```
<?xml version="1.0" encoding="IBM500"?><EMP
ID="000010">HAAS</EMP><FIRST>CHRISTINE<!--FIRSTNME EXAMPLE--></ FIRST><!--THIS IS
JUST A SIMPLE EXAMPLE-->
```

XMLFOREST EASE OF USE

```
SELECT XMLELEMENT (NAME ROOT, XMLFOREST (
   EUOBTRNS_FLOW_ID,
   EUOBTRNS_NBR))
   FROM EU.EUOB31TTRNS
```

```
<?xml version="1.0"
encoding="IBM500"?><ROOT><EUOBTRNS_FLOW_ID>82</EUOBTRNS_FLOW_ID><EUOBTRNS
_NBR>12345AB6CD7EFG</EUOBTRNS_NBR></ROOT>
<?xml version="1.0"
encoding="IBM500"?><ROOT><EUOBTRNS_FLOW_ID>70</EUOBTRNS_FLOW_ID><EUOBTRNS
_NBR>12345AB6CD7EFG</EUOBTRNS_NBR></ROOT>
<?xml version="1.0"
encoding="IBM500"?><ROOT><EUOBTRNS_FLOW_ID>97</EUOBTRNS_FLOW_ID><EUOBTRNS
_NBR>12345AB6CD7EFG</EUOBTRNS_NBR></ROOT>
....
```

Returns 1 column containing 1 XML-document per row

XMLFOREST EASE OF USE AND XMLAGG

```
SELECT XMLELEMENT (NAME ROOT, XMLAGG (XMLFOREST (
   EUOBTRNS_FLOW_ID,
   EUOBTRNS_NBR))
   FROM EU.EUOB31TTRNS
```

will generate one XML file looking like :

```
<?xml version="1.0"
encoding="IBM500"?><ROOT><EUOBTRNS_FLOW_ID>82</EUOBTRNS_FLOW_ID><EUOBTRNS
_NBR>12345AB6CD7EFG</EUOBTRNS_NBR><EUOBTRNS_FLOW_ID>70</EUOBTRNS_FLOW_I
D>...
</ROOT>
```

XMLAGG is an XML aggregated function (returning one value for a set of rows)
It will generate one XML file, containing all of the data of the table.

XMLFOREST CHANGING THE TAG NAMES

```
SELECT XMLELEMENT (NAME ROOT, XMLAGG (XMLFOREST (
   EUOBTRNS_FLOW_ID as ID,
   EUOBTRNS_NBR as NBR )))
   FROM EU.EUOB31TTRNS
```

```
<?xml version="1.0"
encoding="IBM500"?><ROOT><ID>82</ID><NBR>12345AB6CD7EFG</NBR><ID>70</ID>...
</ROOT>
```

By adding the AS-clause you can name your tags.

Default is the column name.

53

LAST BUT NOT LEAST : XMLMODIFY

For XMLMODIFY to work XML versioning format must be used !

⇩

Universal tablespace must be used!

```
SELECT XMLQUERY( '/BOOKS/BOOK[TITLE="FEELS LIKE
HOME"]/PRICE'
     PASSING STOCK )   FROM KST.BOOKSTORE1
<?xml version="1.0" encoding="IBM500"?><PRICE>9.85</PRICE>
<?xml version="1.0" encoding="IBM500"?><PRICE>9.85</PRICE>
<?xml version="1.0" encoding="IBM500"?><PRICE>9.85</PRICE>
```

```
UPDATE KST.BOOKSTORE1 SET STOCK = XMLMODIFY
  ('replace value of node /BOOKS/BOOK[TITLE="FEELS LIKE HOME"]/PRICE
    with /BOOKS/BOOK[TITLE="FEELS LIKE HOME"]/PRICE*1.15')
where storeid = 3
```

```
SELECT XMLQUERY( '/BOOKS/BOOK[TITLE="FEELS LIKE
HOME"]/PRICE'
     PASSING STOCK )   FROM KST.BOOKSTORE1
<?xml version="1.0" encoding="IBM500"?><PRICE>9.85</PRICE>
<?xml version="1.0" encoding="IBM500"?><PRICE>9.85</PRICE>
<?xml version="1.0"
encoding="IBM500"?><PRICE>11.3275</PRICE>
```

INFOCURA

WE CARE ABOUT YOUR INFORMATION

KURT STRUYF

KURT.STRUYF@INFOCURA.BE

FOLLOW ME ON TWITTER : @DB2KURTS